Deep Learning **Fall 2019**

Lecture 2: RNN & LSTM & Attention

Lecturer: Thomas Hofmann *Scribes: Yao Zhang*

Definition 2.1 (State space model)**.** *Given observation sequence $x^1, ..., x^s$. Identify hidden activities h with the state of a dynamical system. Discrete time evolution of hidden state space sequence*

$$h^t = F\left(h^{t-1}, x^t, \theta\right), \quad h^0 = 0, \quad t = 1, ..., s \tag{2.1}$$

1. *Markov property: hidden state at time t depends on input of time t as well as previous hidden state*
2. *Time-invariance: state evolution function F is independent of time t*

How should F be chosen?

Definition 2.2 (Recurrent Neural Network)**.** *Linear dynamical system with elementwise non-linearity*

$$\begin{aligned} \overline{F}(h, x, \theta) &= Wh + Ux + b, \quad \theta \ = \ (U, W, b, ...) \\ F &= \sigma \circ \overline{F}, \quad \sigma \in \{logistic, \ \tanh, \ ReFLU, ...\} \end{aligned} \tag{2.2}$$

Optionally produce outputs via

$$y = H(h, \theta), \quad H(h, \theta) \triangleq \sigma(Vh + c), \quad \theta = (..., V, c) \tag{2.3}$$

Unfolding of recurrency. Recurrent networks: feeding back activities (with time delays). Unfold computational graph over time (also called unrolling)

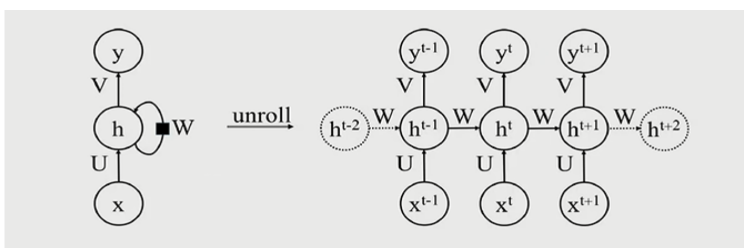


Figure 2.1: Unfolding of recurrency

Lossy memorization: what does a recurrent network (RNN) do?

1. hidden state can be thought of as a noisy memory or a noisy data summary.
2. learn to memorize relevant aspects of partial observation sequence:

$$\left(x^1, \cdots, x^{t-1}\right) \mapsto h^t \tag{2.4}$$

3. more powerful than just memorizing fixed-length context.

Feedforward vs. Recurrent networks:

1. for any fixed length s, the unrolled recurrent network corresponds to a feedforward network with s hidden layers
2. however, inputs are processed in sequence and (optionally) outputs are produced in sequence
3. main difference: sharing of parameters between layers – same function F and H at all layers / time steps.

Backpropagation through time:

1. backpropagation is straightforward: propagete derivatives backwards through time
2. parameter sharing leads to sum over t, when dealing with derivatives of weights
3. define shortcut $\dot{\sigma}_i^t \triangleq \sigma' \left(\bar{F}\left(h^{t-1}, x^t\right)\right)$, then

$$\begin{aligned} \frac{\partial \mathcal{R}}{\partial w_{ij}} &= \sum_{t=1}^{s} \frac{\partial \mathcal{R}}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial w_{ij}} = \sum_{t=1}^{s} \frac{\partial \mathcal{R}}{\partial h_i^t} \cdot \dot{\sigma}_i^t \cdot h_j^{t-1} \\ \frac{\partial \mathcal{R}}{\partial u_{ik}} &= \sum_{t=1}^{s} \frac{\partial \mathcal{R}}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial u_{ij}} = \sum_{t=1}^{s} \frac{\partial \mathcal{R}}{\partial h_i^t} \cdot \dot{\sigma}_i^t \cdot x_k^t \end{aligned} \tag{2.5}$$

RNN gradients: RNN where output is produced in last step: $y = y^s$. Remember backpropagation in MLPs:

$$\nabla_x \mathcal{R} = J_{F^1} \cdots J_{F^L} \nabla_y \mathcal{R} \tag{2.6}$$

Shared weights: $F^t = F$, yet evaluated at different points

$$\nabla_{x^t} \mathcal{R} = \left[\prod_{r=t+1}^{s} W^T S\left(h^r\right)\right] \cdot \underbrace{J_H \cdot \nabla_y \mathcal{R}}_{\triangleq z} \tag{2.7}$$

where $S\left(h^r\right) = diag\left(\dot{\sigma}_1^t, ...\dot{\sigma}_n^t\right)$, which is $\leq I$ for $\sigma \in \{logistic,\ tanh,\ ReLU\}$.

Exploding and/or vanishing gradients: spectral norm of matrix which is the largest singular value

$$\|A\|_2 = \max_{x:\|x\|=1} \|Ax\| = \sigma_{\max}\left(A\right) \tag{2.8}$$

Note that $\|AB\|_2 \leqslant \|A\|_2 \cdot \|B\|_2$, hence with $S\left(\cdot\right) \leqslant I$

$$\left\|\prod_{s=t+1}^{s} W^T S\left(h^t\right)\right\|_2 \leqslant \left\|\prod_{s=t+1}^{s} W^T\right\|_2 \leqslant \|W\|_2^{s-t} = \left[\sigma_{\max}\left(W\right)\right]^{s-t} \tag{2.9}$$

If $\sigma_{\max}\left(W\right) < 1$, gradients are vanishing, i.e.

$$\|\nabla_{x^t} R\| \leqslant \sigma_{\max}(W)^{s-t} \cdot \|z\| \overset{(s-t)\to\infty}{\to} 0 \tag{2.10}$$

Conversely, if $\sigma_{\max}\left(W\right) > 1$ gradients may explode. (depends on gradient direction [1]).

Bi-directional recurrent networks: hidden state evolution does not always have to follow direction of time (or causal direction).

Define reverse order sequence

$$g^t = G\left(x^t, g^{t+1}, \theta\right) \tag{2.11}$$

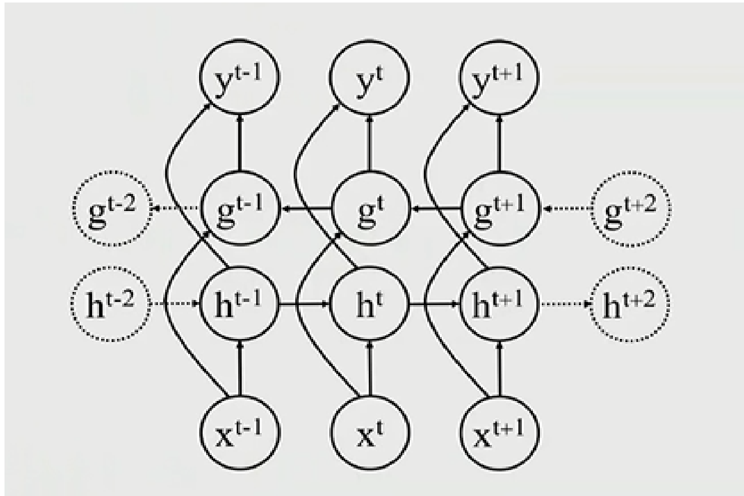


Figure 2.2: hidden state sequences

as model with separate parameters.

Now we can interweave hidden state sequences (see Fig. 2.2). Backpropagation is also bi-directional.

Deep recurrent networks: hierchical hidden state:

$$\begin{aligned} h^{t,1} &= F^1\left(h^{t-1,1}, x^t, \theta\right) \\ h^{t,l} &= F^l\left(h^{t-1,l}, x^t, \theta\right) \quad l = 1, ..., L \end{aligned} \tag{2.12}$$

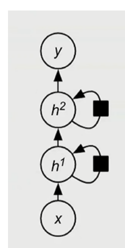


Figure 2.3

Output connected to last hidden layer

$$y^t = H\left(h^{t,L}, \theta\right) \tag{2.13}$$

Can be combined with bi-directionality (how?).

Differentiable memory: long-term dependencies

1. sometimes: important to model long-term dependencies $\Rightarrow$ network needs to memorize features from the distant past
2. recurrent networks: hidden state needs to preserve memory
3. conflicts with short-term fluctuations and vanishing gradients
4. conclusion: difficult to learn long-term dependencies with standard recurrent network
5. popular remedy: gated units

LSTM: overall architecture, Long-Short-Term-Memory: unit for memory management

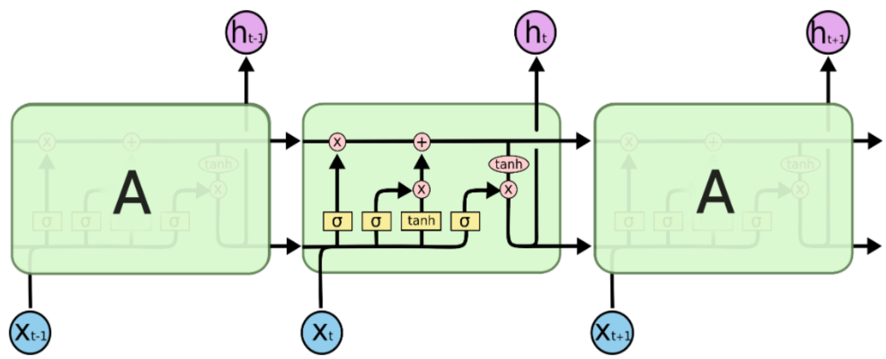


Figure 2.4: The repeating module in an LSTM contains four interacting layers

where

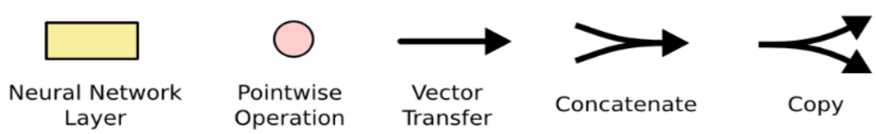
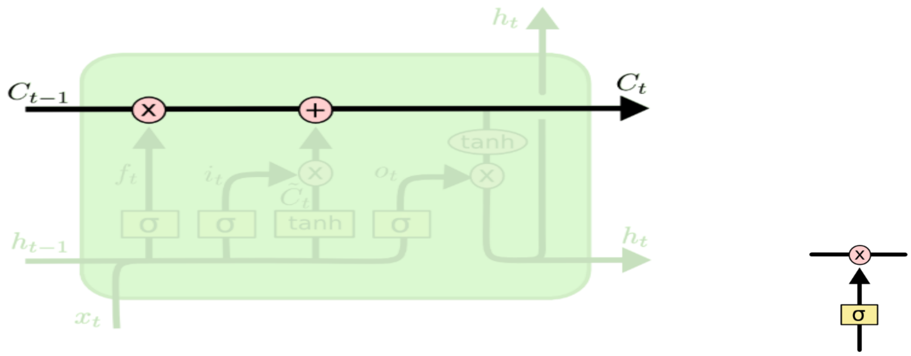


Figure 2.5: from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: flow of information



(a) (b)

Figure 2.6: flow of information

1. information propagates along the chain like on a conveyor belt
2. information can flow unchanged and is only selectively changed (vector addition) by σ-gates

LSTM: forget gate

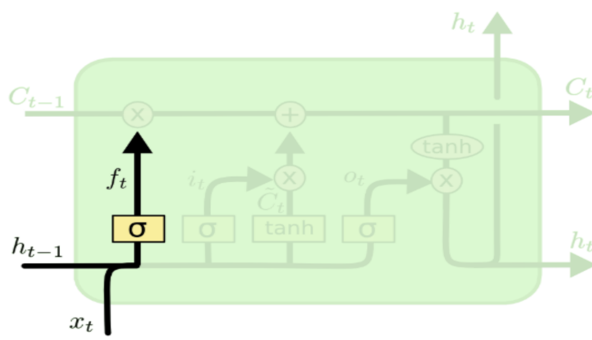


Figure 2.7: forget gate

where

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{2.14}$$

1. keeping or forgetting of stored content?

LSTM: input → memory value where

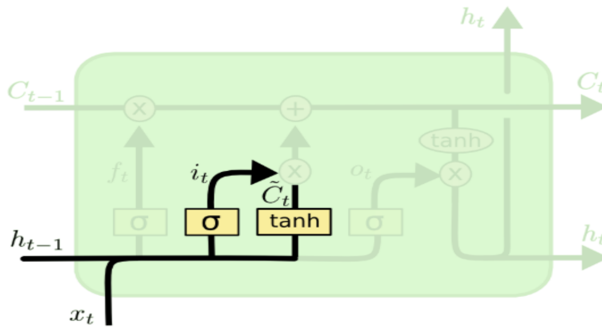


Figure 2.8: input → memory value

$$\begin{aligned} i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\ \widetilde{C}_t &= \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \end{aligned} \tag{2.15}$$

1. preparing new input information to be added to the memory

LSTM: updating memory

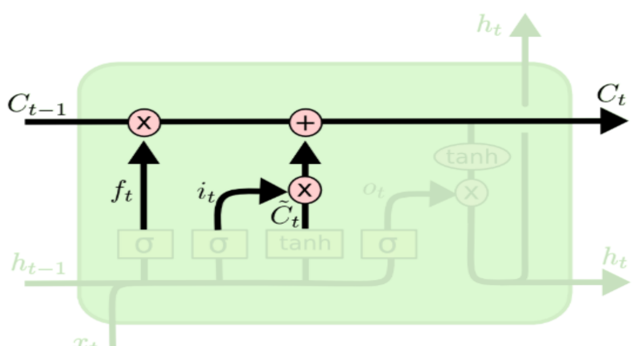


Figure 2.9: updating memory

where

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \tag{2.16}$$

1. combining stored and new information

LSTM: output gate where

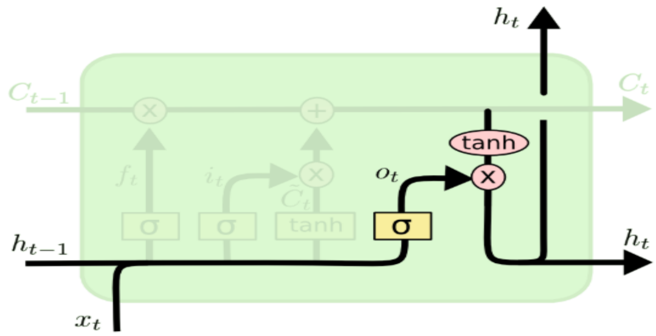


Figure 2.10: output gate

$$
\begin{aligned}
o_t &= \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \\
h_t &= o_t * \tanh\left(C_t\right)
\end{aligned}
\tag{2.17}
$$

1. computing output selectively

LSTM: gate memory units

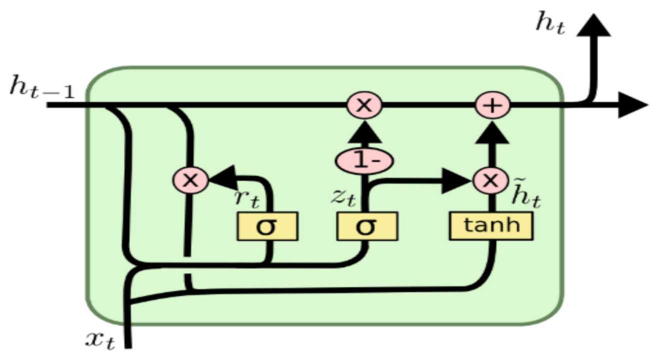


Figure 2.11: gate memory units

where

$$
\begin{aligned}
z_t &= \sigma\left(W_z \cdot \left[h_{t-1}, x_t\right]\right) \\
r_t &= \sigma\left(W_r \cdot \left[h_{t-1}, x_t\right]\right) \\
\widetilde{h}_t &= \tanh\left(W \cdot \left[r_t * h_{t-1}, x_t\right]\right) \\
h_t &= \left(1 - z_t\right) * h_{t-1} + z_t * \widetilde{h}_t
\end{aligned}
\tag{2.18}
$$

1. memory state = output. modification to logic [2]
2. convex combination of old and new information

Gated memory units:

1. GRUs and LSTMs can learn active memory strategies: what to memorize, overwrite and recall when
2. successful use cases:
   (a) handwriting recognition
   (b) speech recognition (also: Google)
   (c) machine translation
   (d) image captioning
3. notoriously difficult to understand what units learn... Resource-hungry. Slow in learning.

Language modeling:

Model	Test Perplexity	Number of Params [billions]
Sigmoid-RNN-2048 (Ji et al., 2015a)	68.3	4.1
Interpolated KN 5-gram, 1.1B n-grams (Chelba et al., 2013)	67.6	1.76
Sparse Non-Negative Matrix LM (Shazeer et al., 2015)	52.9	33
RNN-1024 + MaxEnt 9-gram features (Chelba et al., 2013)	51.3	20
LSTM-512-512	54.1	0.82
LSTM-1024-512	48.2	0.82
LSTM-2048-512	43.7	0.83
LSTM-8192-2048 (No Dropout)	37.9	3.3
LSTM-8192-2048 (50% Dropout)	32.2	3.3
2-Layer LSTM-8192-1024 (BIG LSTM)	30.6	1.8
BIG LSTM+CNN Inputs	**30.0**	**1.04**
BIG LSTM+CNN Inputs + CNN Softmax	39.8	**0.29**
BIG LSTM+CNN Inputs + CNN Softmax + 128-dim correction	35.8	**0.39**
BIG LSTM+CNN Inputs + Char LSTM predictions	47.9	**0.23**

Figure 2.12: Best results of single models on the 1B word benchmark [3]

1. evaluation on corpus w/ 1B words
2. number of parameters can be in the 100Ms or even Bs!
3. ensembles can reduce perplexity to ~ 23 (best result 06/2016)

Sequence to sequence learning:

1. important use of of memory units: sequence to sequence learning. Seminal paper [4]
2. encoder-decoder architecture Encode sequence (e.g. sentence) into vector, decode sequence (e.g. translate) from vector(with autoregressive output feedback)

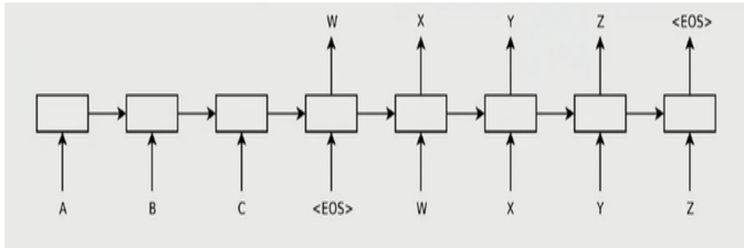


Figure 2.13: encoder-decoder architecture

RNN encoder/decoder: How to make this work? [4]

1. deep LSTMs (multiple layers, e.g. 4)
2. different RNNs for encoding and decoding
3. teacher forcing (maximum likelihood) during training
4. beam search for decoding at test time
5. reverse order of source sequence
6. ensemble-ing
7. $\Rightarrow$ state-of-the art results on WMT benchmarks at the time. Today: use of attention-based models.

Attention Mechanisms:

1. simple way to overcome some challenges of RNN-based memorization: attention mechanism selectively attend to inputs or feature representations computed from inputs.
2. RNNs: learn to encode information relevant for the future.
   vs.
   Attention: select what is relevant from the past in hindsight! Both ideas can be combined

Gating Function:

Definition 2.3 (Softmax Gating Function)**.** *A softmax gating function f_ϕ takes as input a query vector $\xi \in \mathbb{R}^n$ as well as a set of values $x^t \in \mathbb{R}^m$ $(t = 1, ..., s)$ and is defined as*

$$f_\phi\left(\xi, \left(x^1, ..., x^s\right)\right) = \frac{1}{\sum_j e^{\phi(\xi, x^j)}} \begin{pmatrix} e^{\phi(\xi, x^1)} \\ \vdots \\ e^{\phi(\xi, x^s)} \end{pmatrix} \tag{2.19}$$

for some similarity or compatibility function $\phi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$

1. ϕ can often be learned in a black-box manner via MLP
2. simplest choice for $n = m$: $\phi(\xi, x) = \xi^T x$ *(inner product)*
3. every restriction $f_\phi(\xi, \cdot)$ maps to the interior of a simplex

Definition 2.4 (Self-Gated Attention). *Given a query $\xi \in \mathbb{R}^m$ and a set of values $x_i \in \mathbb{R}^n$ $(i = 1, ..., k)$. The self-gated attention is defined as*

$$\underbrace{F\left(\xi, (x_1, ..., x_k)\right)}_{\in\mathbb{R}^k} = \underbrace{[x_1 \; x_2 \; \cdots \; x_k]}_{\in\mathbb{R}^{k\times n}} \cdot \underbrace{f_\phi\left(\xi, (x_1, ..., x_k)\right)}_{\in\mathbb{R}^n} \tag{2.20}$$

where f_ϕ is a gating function.

Seq2seq with Attention: Schematic

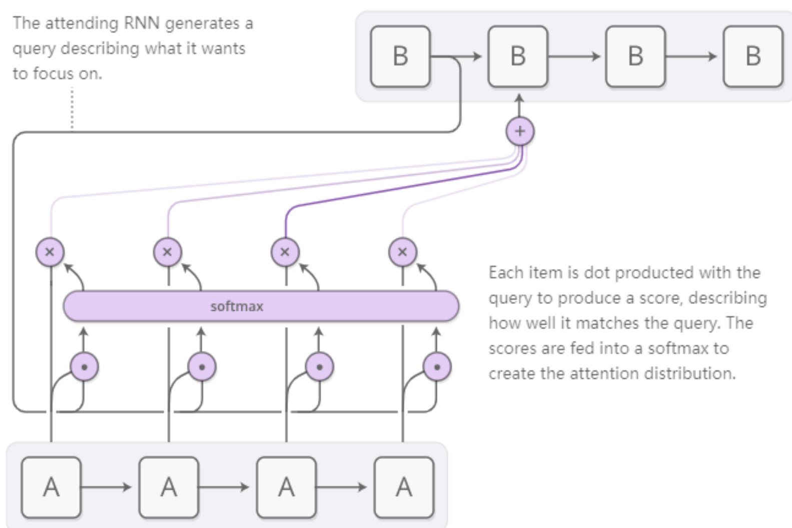


Figure 2.14: from https://distill.pub/2016/augmented-rnns/

Seq2seq with Attention:

1. Attend to the hidden state of the encoding RNN, i.e. values $\left(h_e^1, ..., h_e^s\right)$.
2. Decoding RNN produces query at each time, *i.e.* $\left(\xi^1, ..., \xi^{s'}\right)$.
3. Self-gated attention produces "read-out" z^t from encoder sequence
4. Used ad input to the decoding RNN: $(h_d^t, z^t) \mapsto h_d^{t+1}$

Seq2seq with Attention: MT Example

1. Interpretable attention model (akin to alignments) [5]
2. Bi-directional GRU encoder, left-to-right GRU decoder

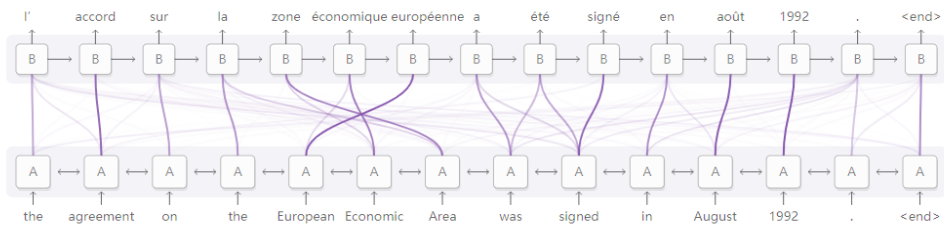


Figure 2.15: from https://distill.pub/2016/augmented-rnns/

Seq2seq with Attention: Speech Recognition

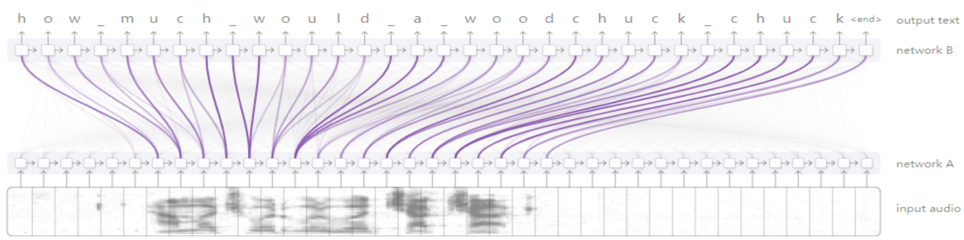


Figure 2.16: from https://distill.pub/2016/augmented-rnns/

1. Listen, Attend and Spell Model [6]
2. Bi-directional, pyramidal LSTM encoder

Memory Networks:

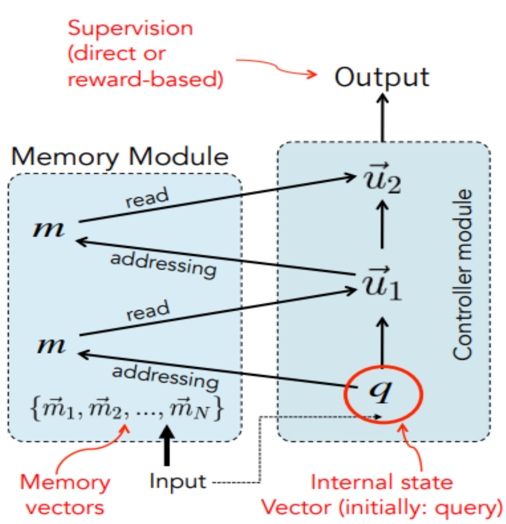


Figure 2.17: from http://www.thespermwhale.com/jaseweston/icml2016/

Definition 2.5 (Key-Value Attention)**.** *Given a query $\xi \in \mathbb{R}^n$, key-value pairs $(x^t,\ z^t) \in \mathbb{R}^n \times \mathbb{R}^m,\ \ t = 1..., s$ and a gating function f. The (n, m)-dimensional key-value attention map is defined as*

$$F\left(\xi, \left(x^1, z^1\right), ..., \left(x^s, z^s\right)\right) = \left[z^1\ z^2\ \cdots\ z^s\right] \cdot f\left(\xi, \left(x^1, ..., x^s\right)\right) \tag{2.21}$$

1. attention weights are computed based on keys
2. produced value is linear (or convex) combination of values
3. keys determine where to look, values determine what features get extracted

Definition 2.6 (Scaled Dot-Product Attention)**.** *The attention map induced by*

$$f\left(\xi, x\right) = \frac{\xi^T x}{\sqrt{n}} \tag{2.22}$$

is called scaled dot-product attention.

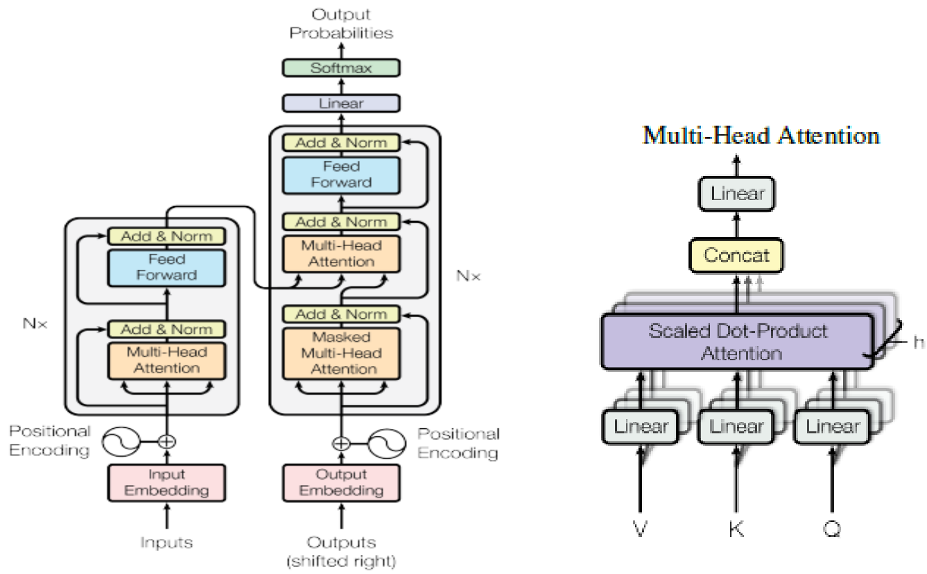
1. simple dot-product similarity between query and key, not necessarily convex (soft-max)
2. motivation for normalization: assume ξ, x are random n-vector with zero mean and unit variances, then
$$E\left[\xi^T x\right] = 0 \quad and \quad E\left[\left(\xi^T x\right)^2\right] = n \tag{2.23}$$

Definition 2.7 (Multi-Headed Attention)**.** *Let $F_j,\ 1 \leqslant j \leqslant r$ be (n, m)-dimensional key-value attention map. An r multi-headed (N, M)-dimensional attention map G is defined as follows:*

$$G\left(\xi, \left(x^t, z^t\right)_{t=1}^{s}\right) = W \begin{bmatrix} F_1\left(W_1^q \xi, \left(W_1^x x^t, W_1^z z^t\right)_{t=1}^{s}\right) \\ \vdots \\ F_1\left(W_1^q \xi, \left(W_r^x x^t, W_r^z z^t\right)_{t=1}^{s}\right) \end{bmatrix} \tag{2.24}$$

1. matrices $W_i^q, W_i^q \in \mathbb{R}^{n \times N}$ and $W_i^z \in \mathbb{R}^{m \times M}$ are linear dimension-reduction matrices (typically: $n < N$and $m < M$)
2. $W \in \mathbb{R}^{M \times r \cdot m}$ adjusts the dimension (typically: reduction)
3. example: design choice in [7]: $r = 8, n = m = 64, N = M = 512$.

Transformer Architecture: Overview



(a) The Transformer model architecture

(b) Multi-Head Attention consists of several attention layers running in parallel

Figure 2.18: Transformer Architecture [7]

Transformer Architecture: Other Design Choices:

1. Fully-connected feedforward networks (specially: ReLU with layer width $512 \mapsto 2048 \mapsto 512$ confer(cf.) 1×1 convolution)
2. Positional encoding: learned or fixed (sine-functions of different frequency)
3. Layer normalization [8] cf. later section on activity re-normalization
4. Skip connections with add (cf. residual layers)

Reading List

[1] R. Pascanu, T. Mikolov and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," *ArXiv*, 2013.

[2] K.Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)* , 2014.

[3] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer and Y. Wu, "Exploring the Limits of Language Modeling," *CoRR*, 2016.

[4] I. Sutskever, O. Vinyals and Q. Le, "Sequence to sequence learning with neural networks," *NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems*, 2014, Vol. 2014, pp. 3104-3112 .

[5] D. Bahdanau, K. Cho and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *3rd International Conference on Learning Representations (ICML)*, 2015.

[6] W. Chan, N. Jaitly, Q. Le and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser and I. Polosukhin, "Attention is All you Need," *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017, Vol. 2017, pp. 5998-6008.

[8] J. Ba, J. Kiros and G. Hinton, "Layer Normalization," *ArXiv*, 2016, Vol.(abs/1607.06450).